Trip report

Automated Software Engineering 2025

Seoul, 2025.11.17 – 11.19

SAL 안주영

1. 개요

11월 17일부터 19일까지 2박 3일간 그랜드 워커힐 호텔에서 열린 ASE 2025에 참석하였다. ASE는 소프트웨어 공학 분야에서 영향력이 높은 학회 중 하나로, 프로그램 분석, 테스팅, 버그 탐지, AI기반 자동화 등의 최신 연구에 대하여 살펴볼 수 있는 좋은 기회였다. 이와 같은 높은 수준의 학회에서 보고 이야기하며 느낀 것들이 연구실 동료들에게 도움이 되고자 하는 바람으로 해당 report를 작성하고자 한다.

2. 연구 및 논문

A. Bug Understanding

i. Agents in the Sandbox: End-to-End Crash Bug Reproduction for Minecraft

해당 연구는 단순히 game상 존재하는 bug를 reproduct하는 것이 아니라, 지속적인 update를 통해 변화되는 프로그램에서 crash bug를 LLM를 활용하여 reproduct할 지에 대하여 논의하기 위해 “Minecraft”라는 대중적이고 꾸준한 update가 진행되는 게임을 통하여 이야기하고자 한다. 기존의 연구에서도 LLM을 통하여 소프트웨어 전반적으로 bug reproduct 분야의 발전이 있었지만, 여전히 runtime에 복합적으로 상화작용이 일어나는 게임 환경상에서는 성공적으로 bug reproduct를 하지 못하였다. 해당 연구를 이를 해결하기 위하여 **step-synthesizer** (S2R)라는 도구를 활용하여 기존 게임에 대한 지식을 기반으로 LLM이 기존의 bug report를 보다 구조화된 단계로 해당 bug reproduction을 설계한다. 이후 **Action Model**(vision-base LLM agent)을 통한 게임 화면 분석을 기반으로 S2R가 생성한 단계 별로 bug reproduction을 수행하게 된다. 게임을 통해 다양한 상호작용이 일어나는 환경에서도 bug를 안정적으로 reproduct하기 위해 LLM을 통하여 다양한 변수들을 통제하여 했다는 점에서 매력적이었다.

ii. Reinforcement Learning for Mutation Operator Selection in Automated Program Repair

해당 연구는 기존의 heuristic-based program repair에서 사용한 uniformly-random한 mutation operation 선택을 통해 생성되는 단점들을 보완하기 위해 강화학습을 사용한고자 한다. 기존의 heuristic기반의 선택 방식들을 종합하여 4가지 mutation 방식과 2가지 reward를 기반으로 강화학습을 진행하여 Defects4J(java benchmark)에서 기존 방식보다 더 유효한 mutant를 생성하는 것을 확인하였다. 하지만 해당 논문에서는 여전히 patch한 bug가 기존방식과 유사하다는 단점과 더불어 단순히 mutate 방식에만 집중하여 다른 변수들(mutation 위치, 사용하는 mutation element...)에 대하여는 고려하지 않았다는 점에서는 아쉬점이 더 큰 연구였다.

B. Fuzzing

i. Algernon: A Flag-Guided Hybrid Fuzzer for Unlocking Hidden Program Paths

해당 연구는 현재 연구중인 분야(hybrid fuzzing)에 속한 논문이기때문에 학회 시작이전부터 매우 관심있게 보았지만, 논문이 on-line상에는 존재하지 않아 궁금증이 많은 연구였다. 해당 연구는 c 프로그램 중 flag를 기반으로 분기문들을 benchmark의 특성을 관찰하여 cve를 더 많이 생성하기 위한 새로운 방식을 제시하였다. FDG(Flag Dependency Graph)를 새롭게 생성하여 오직 Flag에 관여하는 constraint만을 수집하여 smt solver에게 전달하고 입력의 중요도 기준으로 삼았다. 현재 관찰하고 있던 문제점 중 하나라 놀랐지만 다행히 우선 목표하는 benchmark가 다르고, 코드 레벨로는 나오지 않아 아이디어적인 측면에서의 비교만 진행해도 좋을 듯 하다. 왜 기존 최신 hybrid fuzzer와 비교하지 않았는지 의문점이 존재하며, 어떠한 방식으로 constraint를 수집하고 solving시도하였는지, 기존에 solving이 안되는 지점을 어떻게 해결하였는가에 대하여는 추후 메일을 보내 확인 중에 있다.

ii. WingMuzz: BlackBox Testing of IoT Protocols via Two-Dimensional Fuzzing Schedule

해당 연구는 IoT를 대상으로 fuzzing을 통하여 취약점을 검출하는 새로운 방식을 제시한다. IoT 기기의 경우 source code와 firmware가 공개되지 않는 경우가 많아 이전의 연구는 black-box fuzzing을 진행하였다. 하지만 해당 연구에서는 더 좋은 feedback을 주기 위해 wingmate라는 개념을 사용하였다. Wingmate는 target program의 Counterpart open-source code를 기반으로 coverage guided-fuzzing을 진행한다는 것이다. 단순히 grey-box fuzzing이 아닌, 다양한 종류와 version의 open-source code를 기존의 seed queue를 기반으로 평가하여 더 좋은 wingamte를 고르는 과정을 통하여 보다 더 많은 취약점을 탐지 할 수 있게 되었다. 해당 연구를 통해 source-code가 공개되지 않는 목표 프로그램에 대하여서도 feedback을 통해 해당 프로그램을 더 효율적으로 fuzzing할 수 있는 새로운 방식을 알 수 있게 되었다.

iii. Terminator: enabling efficient fuzzing of closed-source GUI programs by automatic coverage-guided termination

해당 연구는 GUI 프로그램을 대상으로 퍼징을 더 효율적으로 정하기 위해 coverage기반으로 실행 Termination을 정하는 방식을 제시하였다. 많은 real-world applications의 경우 input file이 process되는 중에는 종료되지 않기 때문에 효율적으로 퍼징하기 위해서는 강제적으로 종료해야할 필요가 있다. 이전의 연구들은 이를 해결하기 위햇 harness code를 작성했지만, 해당 방식은 목표 프로그램의 구조를 알아야 하며 reverse engineering을 하기 위해 많은 노력이 들어간다. 해당 연구는 이러한 overhaed는 제거하기 위해 execution tree를 수집하여 목표 funciton에 도달하기 위한 optimal termination point를 계산하게 된다. 이후 해당 point에 도달하면 종료하는 Patch를 적용하여 fuzzing을 진행하는 식으로, 기존보다 단일 입력에 대하여 1.4배에서 30배의 boost-up 효과와 더불어 더 많은 cve를 탐지하는 결과를 도출해내었다. 해당 연구에서 유의깊게 살펴볼 사항은 execution tree기반 optimal termination point를 계산하였다는 것이다. 해당 방식을 기반으로 지향성 퍼징에서도 사용할 수 있을 거 같다는 생각이 들었다. 기존의 low-level에서 line 별 혹은 function-level이 아닌 termination point들 사이의 거리를 통하여 더 효율적인 거리 계산이 되지 않을까 하는 생각이 들었다.

iv. TEPHRA: Principled Discovery of Fuzzer Limitations

해당 연구의 경우 호김심에 들었지만 발표 이후 기존 fuzzer들에 대한 평가와 더불어 방향성을 제시해 주고 있다는 생각이 들어 이후 포스터 세션에서 가장 오랬동안 이야기한 주제이다. 요지는 매우 단순하다, 기존 연구들은 coverage 혹은 found bug의 증가율을 기반으로 성능을 측정하게 되는데, heuristic 기반 Fuzzing을 counter하기 위해 “obstacles”을 프로그램에 삽입하게 된다. 해당 연구는 이러한 Obstacle들을 체계적으로 구현 / 실험을 통해 기존의 fuzzer들이 어떠한 obstacle을 해결하지 못하는지, 즉 어떠한 heuristic들을 잘 생성하지 못하는지에 대하여 연구하였다. Type, Var, Condion이렇게 3개로 구성된 Obstacle들을 depth에 따라 만들고, 실험을 진행하여 실험한 결과 Floating-point 또는 character string을 사용한 조건문은 모든 fuzzer들이 뚫기 힘들어 한다는 것을 확인하였다. 또한 unsigned보다 signed int들에 대한 조건문이 더 뚫기 어려웠으며, 8/16 bit integer들에 대하여 heuristic들이 별로 없다는 것을 알아내었다. 위 obstacle중 하나만 있더라도 fuzzer의 전반적인 성능이 저하되기 때문에 해당 연구는 앞으로 fuzzer들이 어떠한 방향을 가지가야 하는지 더불어, 퍼저들의 성능을 평가하는 보다 발전된 methodology를 제시하였다는 점에서 매력적인 연구라 생각된다.

C. Formal method & Verification

i. Programmers’ Visual Attention on Function Call Graphs During Code Summarization

해당 연구는 우연히 듣게 된 발표이지만, 매우 흥미롭게 들어서 후에 진행되는 포스터 세션에서도 오랫동안 이야기를 나눈 연구이다. 해당 논문의 가장 큰 요지는 사람이 코드를 정리할 때에 어떤 부분을 관찰하는 것이 제일 도움이 많이 되는지를 파악하기 위해 eye-tracker를 통해 요약 quality / 시간 대비 어느 부분(caller/callee)에 집중하여 보았는 지를 파악하여 정리한 것이다. 결론은 흥미롭게도 callee즉 code 파악의 중심이 되는 지점을 기준으로 해당 함수를 호출한 부분(caller)과 해당 함수가 호출한 부분(callee)로 나누어 보았을 callee 부분에 집중한 시간이 많아 질 수록 오히려 summery의 quality가 내려갔으며, caller 부분에 집중한 것은 코드 요약에 별 상관관계가 없다는 것이다. 실험적으로 더 세부적으로 볼 수록 오히려 이해도가 떨어지는 현상을 발견 것이 매력적 연구이다.

ii. Faster Runtime Verification during Testing via Feedback-Guided Selective Monitoring

해당 연구에서는 spec을 검증하기 위해 진행하는 Runtime Verification(RV)를 향상시키기 위해 reinforcement Learning을 접목한 방식을 제시한다. 기존 방식들의 경우 monitor들이 과도하게 중복적으로 생성되어 쓸모없는 Testing이 많이 일어난다는 점을 관찰하여 이를 해결하기 위해 선택적으로 Monitor를 생성하게 된다. 이를 위해 이전의 중복적으로 생성된 monitor에 대한 정보와 event들에 대한 정보를 통한 MAB 강화학습으로 앞으로 생성할 montior를 판단하게 된다. 이러한 방식을 통하여 SOTA대비 20~500배의 boost-up을 가져왔으며 동시에 99%의 violation을 여전히 잡는 것을 보여 적은 cost대비 높은 성능을 가지는 것을 확인하였다.

D. Security

i. Should We Evaluate LLM Based Security Analysis Approaches on Open Source Systems

해당 연구는 사람들이 LLM이 open-source의 취약점을 잘 잡는 것이 실제 성능과 어느 정도 연관성을 가지는 지에 대한 연구를 진행하였다. 기존의 연구들 또한 LLM을 open-source의 취약점을 감지하는 데 사용하지만, 이미 학습으로 사용된 정보(정답)을 통해 취약점을 감지한 것인지 확실하지 않기 때문에 “성능을 신뢰할 수 있느냐”를 실험을 통해 증명하고자 하였다. 이를 위해 open-source / close-source(사내 코드)을 여러 ai model들에게 실험하여 탐지한 취약점의 개수 / 정확도를 판별하였다. 실험한 결과, open-souce의 F1 score가 closed-source대비 20%정도 상위하는 것을 발견하였다. 이는 위에서 걱정한 지점이 실제로 다양한 ai model들에게서 공통적으로 나타난다는 것이다. 해당 관찰결과는 흥미롭지만 여전히 몇가지 의문점은 존재한다. 두가지 코드가 공통적으로 사용하는 library가 얼마나 되는지, program size와 같은 다양한 변수들로 인해서도 성능의 차이가 나기 때문에, 해당 연구를 시작으로 더 다양한 방면으로 후속 연구가 지속 될 수 있는 흥미로운 연구라고 생각된다.

3.연구 동향

이번 ASE에서 50%가 넘는 비율이 LLM/Gen AI 관련한 논문인 것으로 느껴졌다. 물론 SE학회가 가지는 특성으로 인해 해당 비율이 높은 것도 존재하지만, 여러 분야에서 성능의 획기적 상승의 한 주춧돌을 담당하는 것이 AI인 것은 연구적인 측면에서 부정 할 수 없다는 것이 본 학회를 통해 다시한번 느껴졌다. 또한 퍼징 분야에서의 경우 점점 target program의 범위가 넓어지는 것으로 보인다. 본 학회에서 c보다 Java대상 퍼징 논문이 더 많아지는 것과 더불어, IoT와 같은 기존에 존재하지 않던 새로운 분야에 대한 퍼징도 다양하게 시도되어 좀더 다양한 범위에 대하여 시야가 넓어진 계기가 되었다.

4. 개선점

본 학회에서 가장 아쉬운 점이라면 한번에 너무나 많은 발표와 짧은 발표시간인 것 같다. 물론 분야에 따라 여러 부분으로 나누기도 하였지만, 그럼에도 불구하고 한번에 4~5가지 분야의 발표가 진행되다 보니 발표 하나(10분)마다 여러 발표장을 돌아다니다 보니 아쉬운 경험으로 남을 수 밖에 없었다. 발표의 질 또한 포스터 준비와 더불어 발표준비까지 하다 보니 포스터 세션이전에 진행되는 발표를 너무 간략하게 진행하거나, 포스터에 발표자료를 오려다 붙이는 경우도 존재하다 보니 관심이 있던 연구 주제도 이해가 힘들었던 경험도 존재한다.

5.마치며

첫 SE학회로서, 두번째 국제 학회 참여로서 다양한 발표(keynote, research ..)를 잘 이해할 수 있을지 걱정 되었지만, 다행히 이전 학회보다 더 다양한 분야에 대한 이해와 더불어 더 다양한 연구자들과 자유롭게 토론하며 의견을 나눌 수 있는 좋은 기회가 되었다. 또한 이전에 참석한 PL 학회보다 넓고 생소한 분야에 대한 연구가 더 많이 포함되어 있어 현재 연구중인 분야와 더불어 새롭게 연구 하고 싶은 분야에 대하여 폭넓은 탐색이 가능했던 좋은 기회가 이번 학회가 되었다.

6.사진